Arrays in Java

- An array is a collection of similar data type values.
- Syntax: `datatype[] variable_name = new datatype[size];`

OR

```
datatype[] variable_name;

variable_name = new datatype[size];
```

OR

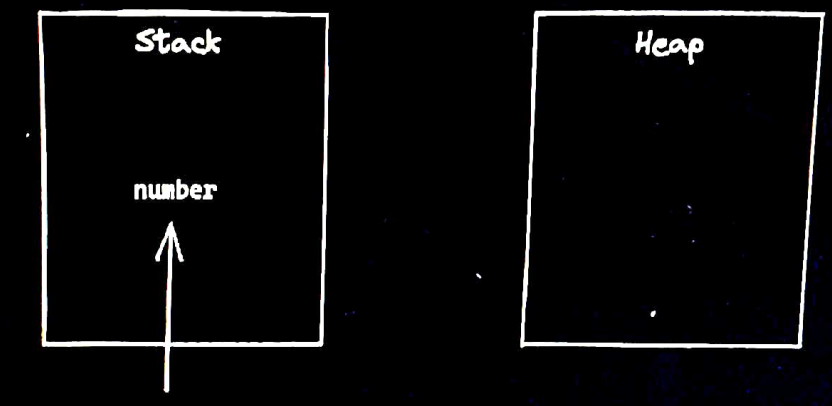
```
datatype[] variable_name = {value1, value2, value3, . . ., valueN};
```

Understanding the Syntax

- `datatype[] variable_name;`

This step will initialize the variable and it will get initialized in the stack during compile time.

Example: `int[] number;`

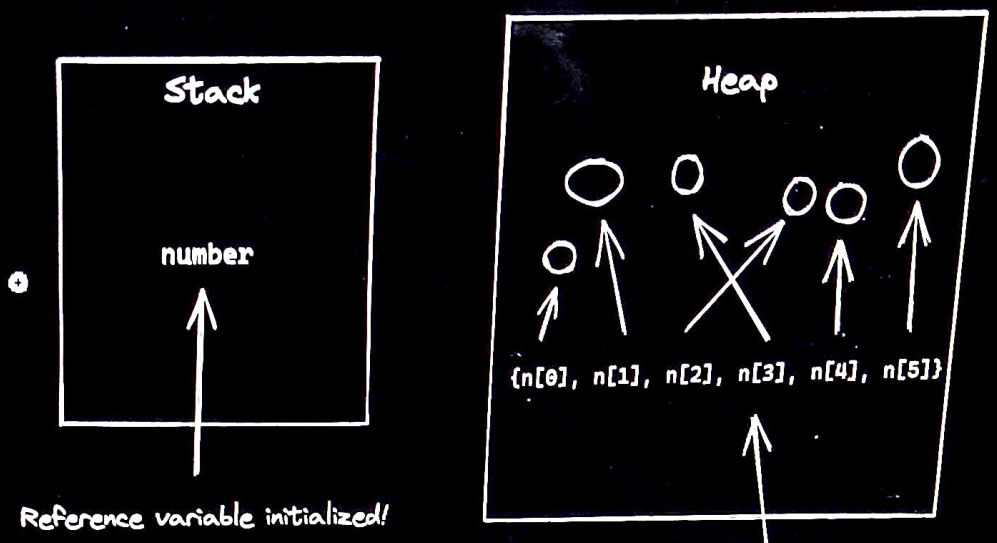


◇ `variable_name = new datatype[size];`

In this step, a new object will be created in the heap memory during runtime.

The 'new' keyword is used to create a new object.

Example: `n = new int[6];`



Reference variable initialized!

A new object is created with a size of six reference variables

If these reference variables have nothing to point to, they will return 'null' when called.

Arrays in Java

2D Arrays

- A 2D array can be visualized as a matrix. Let's understand how?

- → First of all, let's take a 1D array like this,

```
int[] num = {2, 5, 6, 9, 3};
```

→ Now, write this array vertically like this,

```
int[] num = {2,
             5,
             6,
             9,
             3};
```

rows = 5

→ Then, replace each element with an array like this:

```
int[][] num = {
                {1, 2, 3, 4},
                {5, 6, 7, 8},
                {9, 10, 11, 12},
                {13, 14, 15, 16},
                {17, 18, 19, 20}
              };
```

(2D array)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

rows = 5

columns = 4

total elements = 5*4 = 20

→ You can also do something like this:

```
int[][] num = {
                {1, 2, 3, 4},
                {5, 6, 7},
                {8, 9, 10, 11},
                {12, 13},
                {14}
              };
```

(dynamic array)

rows = 5
columns = dynamic

◊ Syntax: `datatype[][] variable_name = new datatype[row_size][column_size];`

OR

```
datatype[][] variable_name;
variable_name = new datatype[row_size][column_size];
```

OR

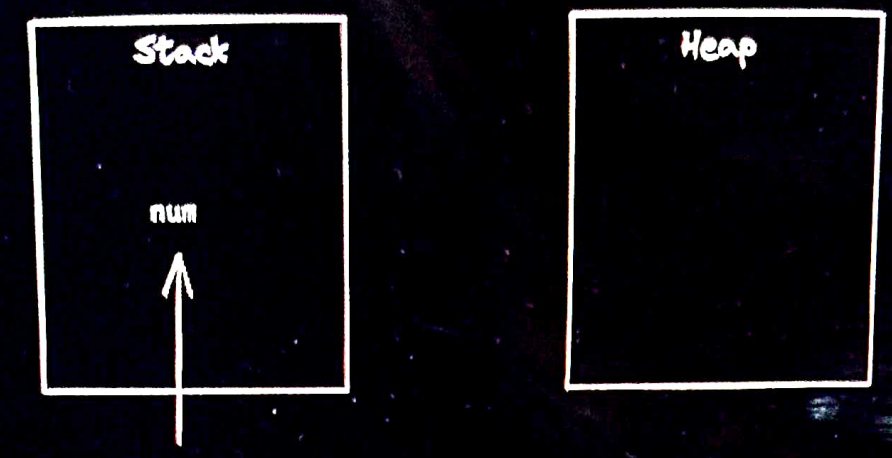
```
datatype[][] variable_name = {{array1}, {array2}, {array3}, . . ., {arrayN}};
```

Understanding the Syntax

◊ `datatype[][] variable_name;`

This step will declare the variable and it will be declared in the stack during compile time.

Example: int[][] num;



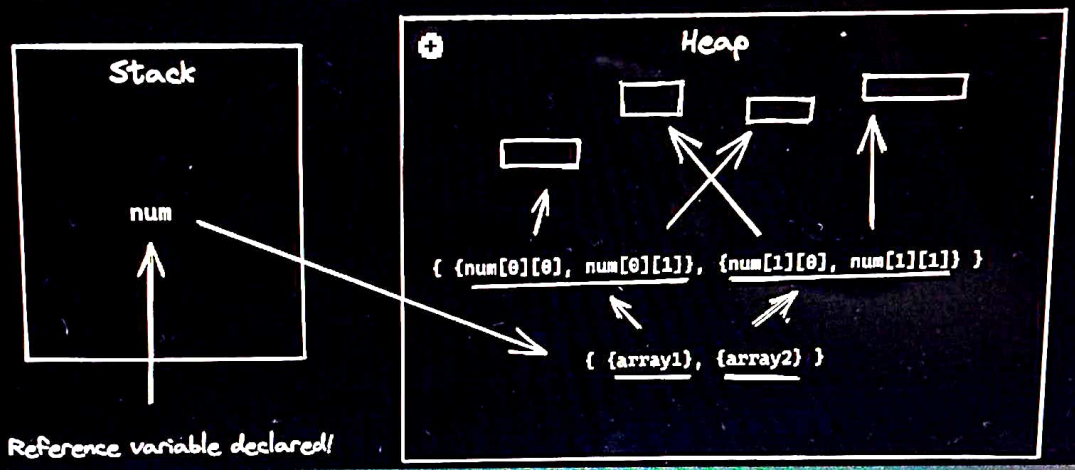
Reference variable declared!

◇ variable_name = new datatype[row_size][column_size];

In this step, a new object will be created/initialized in the heap memory during runtime.

The 'new' keyword is used to create a new object.

Example: num = new int[2][2];



Reference variable declared!

ArrayList in Java

First of all let's understand the difference between Arrays and ArrayList

Array

- An Array has a fixed length or size.
- An Array can be created using both primitive datatypes as well as non-primitive datatypes.

ArrayList

- An ArrayList can contain as many elements as you want even though an initial size is specified.
- An ArrayList cannot be created using primitive datatypes. It can only be created using objects or wrapper classes.

What is an ArrayList?

Let's say you don't want to set a fixed size or length of an array because either you don't know how many elements you may add or you may want to update the size in the future.

What will you do? That's where ArrayLists come into action.

Syntax

```
ArrayList<Integer> list = new ArrayList<Integer>(10);
```

- `ArrayList` → Class
- `<Integer>` → Datatype (Wrapper Class) [not primitive]
- `list` → reference variable
- `new` → creates a new object
- `ArrayList<Integer>(10)` → ArrayList constructor
- `10` → Initial size

Some methods

- add() :- Adds a new element to the ArrayList
- set(index, value) :- Updates an existing value for a specified index
- get(index) :- Used to retrieve an existing value for a specified index

Internal Working

Actually the size of the ArrayList is fixed but not permanently.
It can change according to the input you provide.

Example:-

- You provide the input for the first time and the initial size is set to 5

- Then, you decide that you want to add another four elements to the ArrayList.
But we don't have enough space. So, what will Java do?

The answer is, it will create a new list with a new size (it depends) enough to accommodate new elements.

Then, it will copy the old list to the new list and will delete the old list...

It might look like this: